SpaceOps Conferences
28 May - 1 June 2018, 2018, Marseille, France
2018 SpaceOps Conference

10.2514/6.2018-2558

Applications of Deep Learning Neural Networks to Satellite Telemetry Monitoring

Corey O'Meara *, Leonard Schlag†, Martin Wickler‡
Deutsches Zentrum für Luft- und Raumfahrt e. V., German Aerospace Center
Münchener Straße 20, 82234 Weßling, Germany

As part of the Automated Telemetry Health Monitoring System (ATHMoS) being developed at GSOC, we performed an investigation into potential applications of artificial neural networks to our existing health monitoring system. In the end, we have created an experimental module which uses several deep learning neural networks to augment our existing data analysis algorithms. The module accomplishes three things; automatic feature extraction, anomaly detection, and telemetry prediction. Automatic feature extraction is used to determine numerical values which represent a time window of a single parameters telemetry data, then, using these abstract numeric values we augment our existing so-called feature vectors used in our ATHMoS anomaly detection system to provide additional information in the anomaly detection.

Additionally, we use the same type of neural network to perform anomaly detection on each telemetry parameter in order to take an ensemble machine learning approach to detecting anomalies in telemetry. By combining the results of the neural network with our existing (non-neural network) anomaly detection algorithms, we can provide a more robust classification of whether or not new data should be flagged as anomalous.

Lastly, we've created a neural network which given the most recent orbit data, can predict the general behaviour of a telemetry parameter over the next four and a half hours. Thus, if the prediction we obtain is off from the usual nominal behaviour of the telemetry parameter, we flag it and label it as a potential future anomaly - thereby performing anomaly prediction.

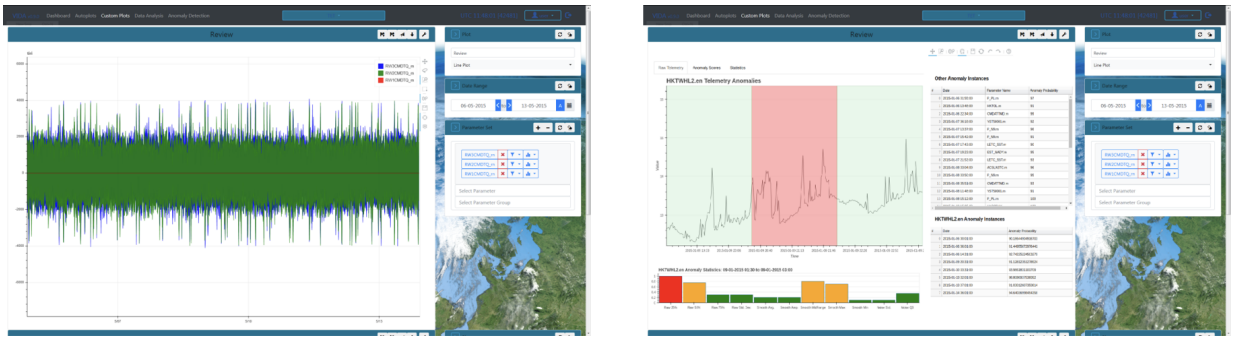
Adding these experimental neural network capabilities to work concomitantly with the already existing anomaly detection modules which ATHMoS is comprised of, has already shown to be beneficial by providing new insights into the data as well as offering a more robust approach to anomaly detection via ensemble machine learning. Here we present an overview of a year long study into applying neural networks to telemetry monitoring and discuss in detail the way in which the three applications described above are being added to the current anomaly detection system at GSOC.

I. Introduction

To efficiently monitor and analyze satellite telemetry, the modular data analysis framework Automated Telemetry Health Monitoring System (ATHMoS) has been developed by the Data Mining team at the German Space Operation Center (GSOC). ATHMoS uses novel techniques in machine learning to analyze our satellite telemetry in order to distill insights into their current state-of-health. The system is able to automatically detect and analyze new behavior in the telemetry data and identify anomalies, in which case, the appropriate subsystem engineers can be notified so that they can determine what actions to take next. This anomaly detection method makes use of historic data (on-going mission lifetime data) as well as the most recent data in order to detect whether anomalous behavior occurred in the recent data e.g. current dump data for a LEO satellite. The idea of this approach is that the system can detect subtle changes in recent telemetry data, which left unchecked, can potentially lead to a dangerous component failure.

In this paper, we present a new approach to telemetry analysis for space operations – the application of artificial neural networks (deep learning). We have conducted research into finding several novel applications to augment ATHMoS and will discuss each of the applications, our findings, as well as future work in completing the integration into ATHMoS.

*Mission Planning System Engineer, Mission Operations Department, German Space Operations Center, corey.omeara@dlr.de
†MCDS (Mission Control and Data handling System Engineer) Mission Operations Engineer, Mission Operations Department, German Space Operations Center
‡Deputy Head of Mission Operations Department, German Space Operations Center



(a) Custom Telemetry Plots

(b) Anomaly Detection Analysis

Fig. 1 (a) A screenshot of the ViDA web frontend of ATHMoS displaying telemetry plotting capabilities including parameter selection, value filtering and summary statistics. (b) A second screenshot of the ViDA web frontend displaying one of the Anomaly Detection pages which shows the anomaly interval (red background), nominal intervals (green background) as well as other system statistics.

II. ATHMoS

ATHMoS is a multi-mission distributed software application which is composed of several modular independently deployable running services (programs) [1]. The services are deployed and ran on a GSOC internal server cluster and can be scaled individually depending on server load in our multi-mission environment. Currently, we are supporting 4 LEO satellite missions during an ongoing test-phase and will gradually scale up to monitor telemetry for all upcoming LEO missions and GEO missions. The system adopts the Service Oriented Architecture pattern, and hence allows for new services and programs to be deployed to the cluster without taking down the whole system, a necessary requirement for operational use. Furthermore, due to the software architectural design, we are able to always maintain multiple instances of each of the services which the system is composed of for system redundancy.

The ATHMoS Infrastructure is composed of three components: the ATHMoS Service Cluster, Telemetry Database Cluster and the Spark Cluster. The ATHMoS Service Cluster is the place where all the separate services are deployed in Docker containers, the Telemetry Database Cluster is a 2-node Apache Cassandra [2] distributed database which stores all the processed telemetry and data mined information, and the Spark Cluster [3] is a 16-core 5-node server cluster which is used to perform big data computations.

Right now, ATHMoS is composed of the following services:

1) Filewatcher Service
2) CSVFileParser Service
3) TelemetryDB Writer Service
4) ATHMoS Core Service
5) ATHMoS Orchestrator Service
6) ViDA Web frontend HTTP Server
7) Deep Learning Module (Experimental)

The Filewatcher Service is connected to the 'Offline Processing' chain which processes newly received telemetry packets and stores comma-separated-value (CSV) files to a file server for long-term storage. Once newly created CSV file have been created with the downlinked satellite telemetry, the Filewatcher notifies the CSVFileParser Service to the location and mission type of the new data to be imported and analysed in ATHMoS. The CSVFileParser Service reads the data files and converts them to a special data type which is used by Apache Spark and then sent via HTTP to the TelemetryDB Writer Service where it is then written to the TelemetryDB. Upon successful data storage, the ATHMoS Orchestrator Service then starts the job of getting the ATHMoS Core Service to perform the various data analysis tasks such as feature vector calculation, automated nominal data creation via custom clustering algorithms as well as anomaly detection.

The ViDA Web frontend HTTP Service is our custom developed website for ATHMoS which allows engineers to monitor their subsystems via plotting of arbitrary telemetry data and viewing the anomaly detection output/statistics created by ATHMoS. Screenshots of ViDA are provided in Figure 1.

Lastly, the Deep Learning Module is the topic of this paper. After a year long investigation into possible applications

and integrability, the module is currently being added to augment the existing ATHMoS Core Service by providing an alternative method and approach to analyse telemetry data. Just as the Core Service runs automatically upon new data storage to the TelemetryDB cluster, this module is triggered to run its analysis on the new data and then writes the various results to the TelemetryDB. The module is written in the Python programming language and uses the Keras library [4] and the Google developed Tensorflow framework [5] for creating the neural networks. Before discussing the various uses of this module, we must first introduce some basic concepts which the ATHMoS Core Service uses in order to understand how the neural network module acts as an addition to the system.

A. ATHMoS Core Algorithms

The core functionality of ATHMoS can be summarized as:

1) Feature Vector Creation: Given past data, determine numeric values which describe an interval of a telemetry parameter
2) Automatic Nominal Telemetry Filtering: Perform clustering and outlier filtering on the feature vectors of past data to determine the set of nominal telemetry which shall be compared to new data
3) Anomaly Detection: When new data exists, we compare it to the previously calculated Nominal Telemetry set. If the new data does not belong to the same set, we flag it as an anomaly.

For a detailed summary of how these steps work and which custom designed algorithms we developed at GSOC specifically for telemetry data, see our related works [1] and [6].

III. Introduction to Artificial Neural Networks

Artificial Neural Networks (hereby referred to as neural networks) are machine learning algorithms which recently have seen a surge in research and industry applications due to the boom of big data and increasing relevance of artificial intelligence and machine learning [7], [8]. Neural networks are connected layers of 'nodes', each of which have several input data values and a single duplicated output which depends on the multiple values which are given to it as input. The general structure of a simple artificial neuron is shown in Figure 2. By creating a sequence of such nodes, one can create multiple layers in which each subsequent node depends on the output of its previous nodes. Deep learning is an umbrella term which is used to describe the machine learning subfield which consists of neural networks which are 'deep' in the sense that they have several sequential hidden/middle layers (see Figure 3).

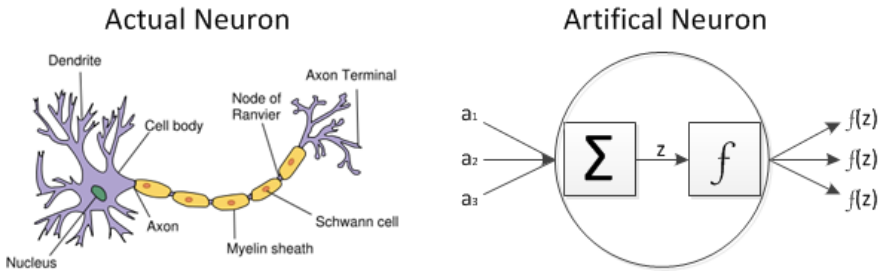


Fig. 2 Left shows the general structure of a neuron cell where multiple input signals are sent as input to the nucleus via the dendrites, and then some modified signal is propagated down through the Nodes of Ranvier and output through to the Axon Terminals. The diagram on the right is a simplistic modelling view of an artificial neuron. As input, there are 3 values which are then summed together by some weighted linear summation and then passed to a so-called activation function f. The output of this function is duplicated and sent through 3 output channels, each of which go to further nodes in the artificial neural network. This is one example of a neural network node, however, there exists more complicated nodes such as Long-Short-Term-Memory nodes (LSTM) [9], which are those that are mainly used in this paper.

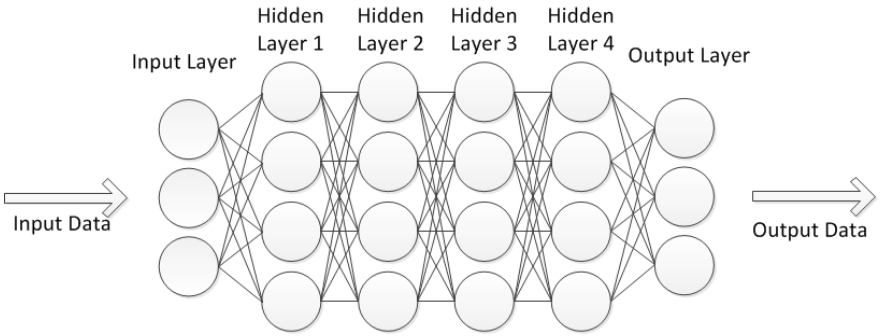


Fig. 3 A multi-layer neural network with 3 input, 3 output nodes, and 4 hidden layers consisting of 4 nodes each. Note that each hidden layer node is connected to the output of all nodes in the previous layer.

In the node diagram in Figure 2, let a_1, a_2, a_3 be input values (numbers) to the neuron and let w_1, w_2, w_3 be some initial weighting applied to each of the components. Then the first thing that is computed inside the node is a weighted sum of the inputs and add some additional bias value b (these weights and biases are initially determined randomly for each neuron). Define this summation as $z = a_1w_1 + a_2w_2 + a_3w_3 + b$. The node applies an activation function f to z in order to determine what the output of this single neuron will be. The activation function is usually one of several standard ones such as the linear, hyperbolic tangent, or sigmoid function. For example, the sigmoid function is given by

$$\sigma(z) = \frac{1}{1 + \exp(-z)}, \tag{1}$$

so that the output of the neuron is within the range $[0, 1]$ as depicted in Figure 4. In fact, the use of such a non-linear function such as the sigmoid function implies that the final output of the total neural network is also non-linear and hence it is able to learn non-linear patterns in the input data. This is very important for our applications since most telemetry data is highly non-linear.

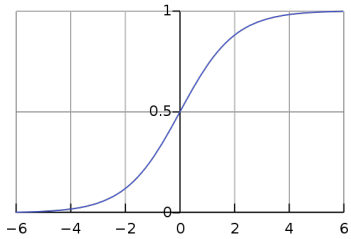


Fig. 4 An example of a the sigmoid activation function used in a simple artificial neuron. Note that the output of the function is a number within the range $[0, 1]$.

The main idea behind a neural network is to create some architecture/pattern of a number of nodes in each layer of the network and then provide a set of input 'training' data, along with a predetermined output data. We start the network learning which consists of cycling through all the input data and compares the end result to the predetermined output data. This training mode comparing the acquired output with the desired output. After one "forward pass", of input data, the network performs a so-called "backward-pass" which then updates updates each i^{th} nodes internal input weight factors w_n^i (for n input values) and bias coefficients b^i usually by some form of gradient descent [10]. The end result is a set of node biases and paired weighting values which have been optimized such that the output of the neural network is as close as possible to the desired output. Usually, letting the network update these internal weights and biases after only one forward and backward pass through all the training data is not sufficient in obtaining accurate output accuracy (determined by some predefined accuracy/measure function). Therefore, we train the network many times on the same input training data and each pass of this training data is referred to as an *epoch*. In this sense, the network automatically learns how to parse the input data in order to obtain a desired output.

Once the network has been trained, it (the biases and weights) can be saved as a model and then used to run against *new* 'test' data which it has never seen before thereby providing genuine new data output. As a simple example, consider

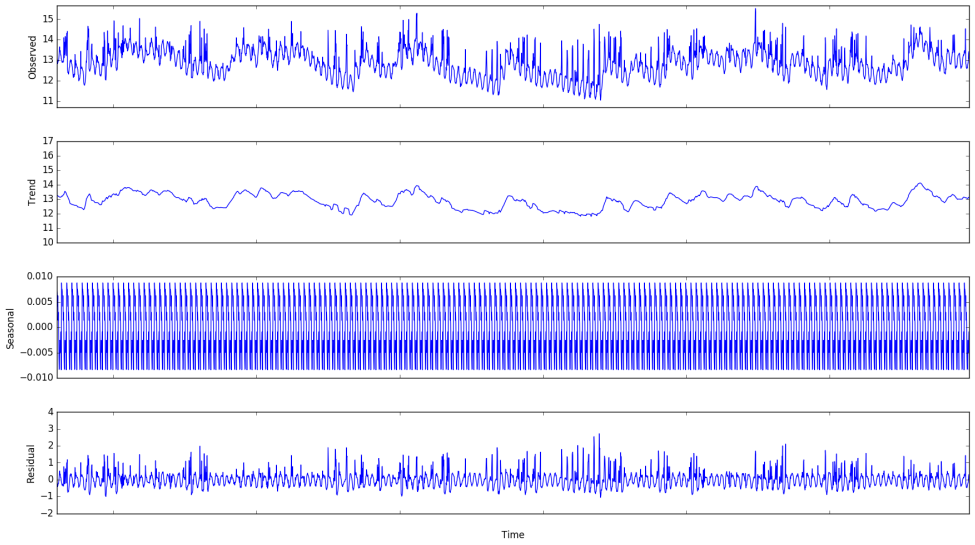


Fig. 5 Decomposing the original Raw Data into its Trend, Seasonal and Residual components (in above descending plot order). For feature vector extraction and anomaly detection, we train the network on only this residual data whereas for time series prediction, we train the network with the Raw/Observed data (since after obtaining a prediction for future data we cannot 're-add' any trend back into the data.)

feeding a neural network a sequence of ordered time series values and suppose the output of the neural network should be the same sequence. This implies the network will attempt to learn an 'identity' function. By feeding the network with many consecutive sequences, each time providing the same time sequence as desired output, it is able to learn a general identity function for this particular set of time series data (up to some reconstruction accuracy). Now, when we have trained a model and feed it a new sequence of time series data, if the reconstruction error is high based on the network output then one can say the input data was different relative to all the input data the model was trained on - thereby indicating that an anomaly occurred within this input time sequence. This simple example is in fact one of the applications we are currently using at GSOC.

The deep learning module currently being implemented at GSOC makes use of a more complicated form of neural network node - the Long-Short-Term-Memory (LSTM) node which is a special type Recurrent Neural Network (RNN) node [9]. Essentially, instead of a single activation function f being applied to the input from the previous nodes output, there are several more activation functions applied, and, the output from the node itself is passed back onto itself so that it can compare network input data with 'the past data'. For time series input, these are typical types of nodes which are used due to the fact that new input from the training dataset implicitly depends on some of the past input data. Additionally, we employ several regularization techniques such as dropout (after each sequence of input data, randomly disconnect a percentage of nodes per layer) [11] and learning rate dampening (changes the step size in stochastic gradient descent as the epoch increases) to obtain our results.

Before discussing the applications of deep learning to our telemetry health monitoring system at GSOC we first should comment on our notation for each section. For each application we will describe the general concept and network architecture which was chosen, e.g. what the training and test inputs are as well as the network configuration we found works best for our use-cases. When discussing input and output data, we will be talking about a single input and output which consists of a sequence of time series data $(x_{t-(B-1)}, ..., x_{t-1}, x_t)$, where x_t is a data point at time t, and B is some fixed 'look-back' integer. Thus, for a look-back number of $B = 18$, the input to the neural network is a time ordered sequence of 18 data points. The total training dataset is then the original time series data which has been split up into overlapping windowed (here would be windows of 18 data points) elements e.g. $[(x_{t-(B-1)}, ..., x_{t-1}, x_t), (x_{(t-B)+2}, ..., x_{t-1}, x_t, x_{t+1}), ...]$.

In terms of hidden layer numbers and nodes, if we used 4 layers each with 150 nodes then the architecture will be described as having a hidden layer layout (150, 150, 150, 150). With these preliminaries set out, we can now discuss our applications.

IV. Automatic Feature Vector Extraction

An autoencoder is a special type of deep neural network which is often used to de-noise input data [12–15]. The general idea is that given some input data, you first enlarge the input into some higher dimensional sparse representation, and then subsequently continue reducing the dimension lower in an iterative fashion until you obtain a compressed representation of the data. This step is known as encoding. Following this encoding, the opposite operation is performed (the decoding) where now you enlarge this compressed data back to its original size and compare this new result with the original data that was input into the neural network. In this sense, the network will learn only what is most important to represent the core attributes/features of the data and automatically ignore such things as anomalies and noise. Figure 6 demonstrates the general structure of such a neural network.

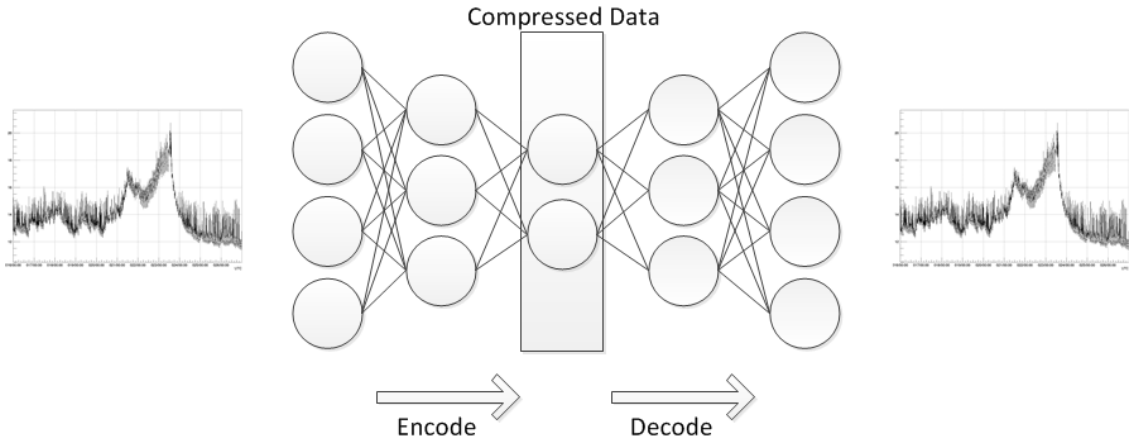
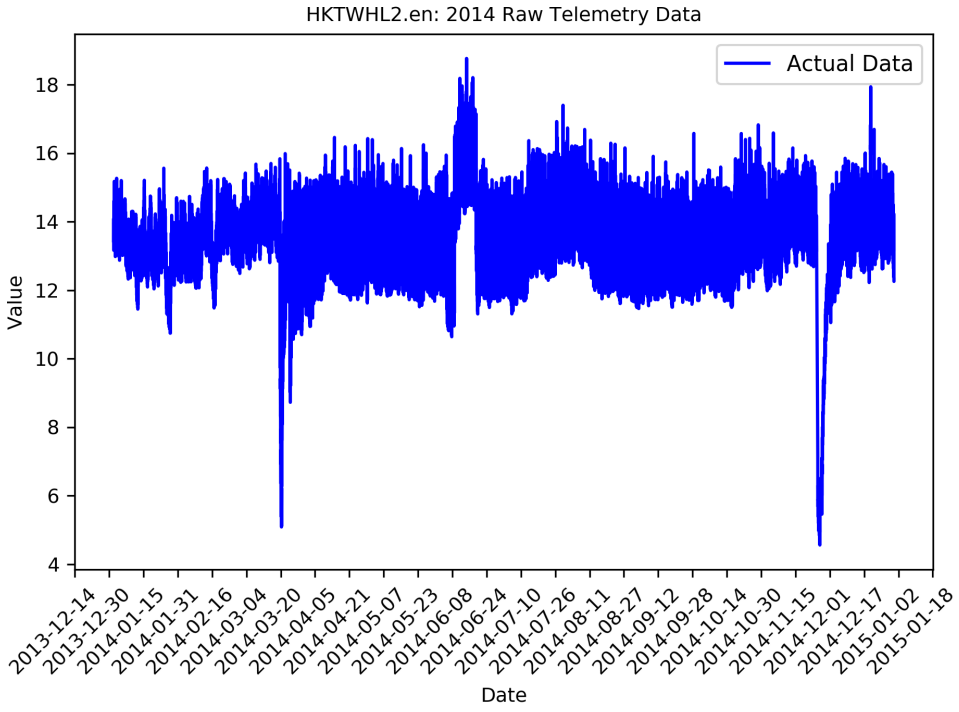


Fig. 6 Autoencoder neural network architecture showing input and output layers with 4 nodes each, 3 hidden layers with nodes 3, 2, and 3, respectively. The first half of the network encodes the data into a lower dimensional vector space (here a 2-d middle layer output), then the second half decodes the data by enlarging this compressed data representation to attempt to reconstruct the original input data.

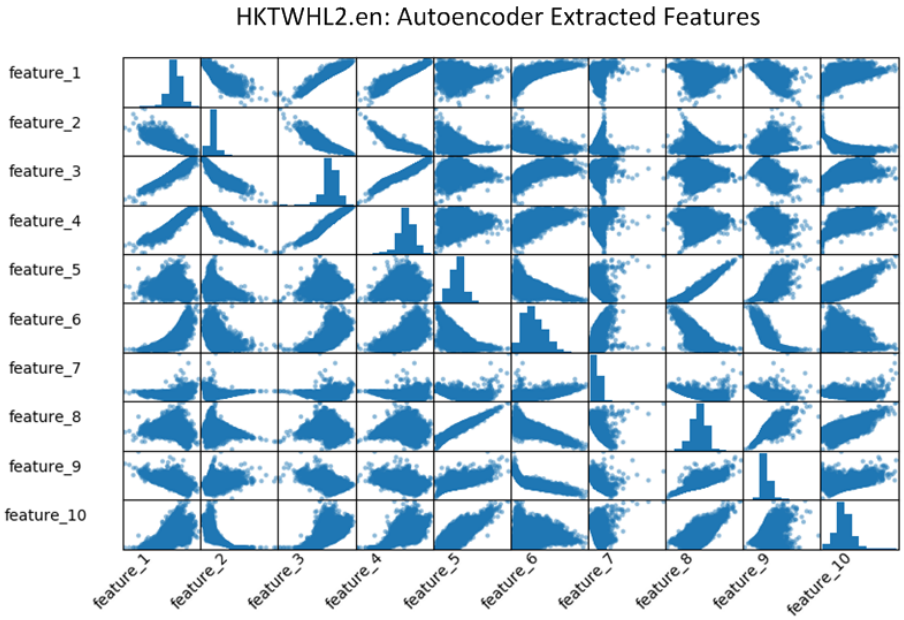
Using an autoencoder neural network, we immediately have two separate applications to ATHMoS; first, the extraction of automatically learned features which represent time intervals of the data which can be used to augment the existing human-engineered/understandable features currently used, and second, directly running anomaly detection. This section will focus on the feature extraction whereas Section V will describe how we use these features to do anomaly detection *only with* the neural network.

The key to obtaining the features is to first train an autoencoder neural network so that it correctly can recreate the original input training data (up to some accuracy measure such as root-mean-squared). For each individual telemetry parameter which now has its own trained network model, we extract the output at *the smallest hidden layer* a.k.a. the compressed representation of the data. In our case, we determined that given an input layer which takes 18 time steps, each 5 minutes apart (hence the input is a subsampled selected of 90 minute orbit data), a smallest hidden layer which contains 10 nodes regularly could reconstruct this original input for the entire training dataset with accuracy $\geq 85\%$.

Thus, extracting the output at this smallest hidden layer we obtain feature vectors composed of features which when plotted against one another result in plots of the form Figure 7. In this figure, we can see the network automatically determined 10 different numeric features that describe the corresponding raw telemetry data. The scatter plots show that the network learned a representation of the data in which 1 large cluster of data points exist as well as several hyper-dimensions where straying elements seem to veer outwards eventually into outliers. By comparing with the raw telemetry data, these data points which are at the border of the center “nominal" cluster in fact correspond to the transition states of the telemetry towards the sharp peaks of anomalous data.



(a) Raw Telemetry Data



(b) Feature Vector Scatter Matrix

Fig. 7 (a) An example of raw telemetry data which was used to calculate auto-extracted features. (b) A scatter matrix plotting automatically generated and extracted features from the autoencoder neural network for the telemetry data shown in (a). The diagonal elements are the histogram distributions of each single feature. Outliers are clearly visible in most of the plots and these correspond to the anomalous sharp peaks in the data in (a).

Now that we can obtain these features that are automatically determined by the network, we can explain how we integrate them into the existing ATHMoS software. Suppose that

$$V_t = (\max, \min, ..., \text{median}) , \tag{2}$$

is a feature vector representing some time interval starting at time t of a certain telemetry parameter and uses some standard features of the data such as the maximum, the minimum, and the median. By feeding in test input into the neural network given by $(x_{t-(B-1)}, ..., x_{t-1}, x_t)$ such that all sequential points are contained within the orbit time which the feature vector V_t represents, then the output of the neural network is

$$V_t^{'} = (\text{feature}_1, \text{feature}_2, ..., \text{feature}_{10}) . \tag{3}$$

The next step we take is to create a new feature vector which is composed of both the human engineered features and those obtained by the neural network

$$\mathbf{V}_t = (\max, \min, ..., \text{median}, \text{feature}_1, \text{feature}_2, ..., \text{feature}_{10}) . \tag{4}$$

This process is repeated to find a complete set of feature vectors which we can run ATHMoS on to detect whether there were any anomalies in the telemetry data (see Figure 8).

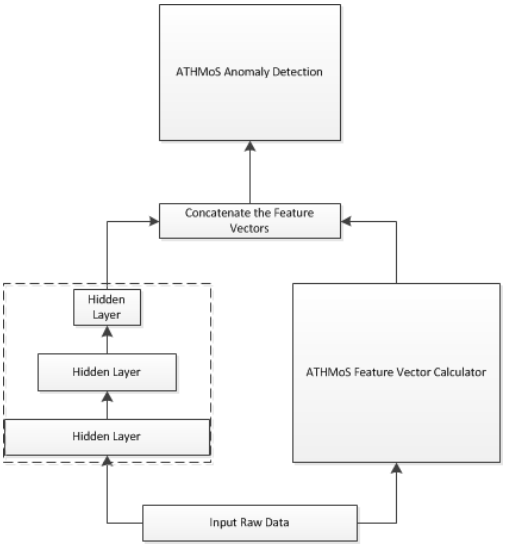
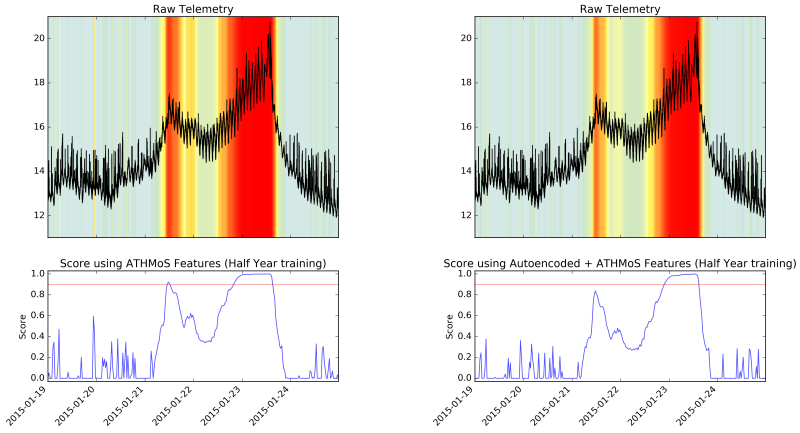


Fig. 8 The process which is being implemented at GSOC where new telemetry data is received and then fed into two different modules, the neural network module for automatic feature vector extraction, and the ATHMoS Feature Vector Calculator. Upon successfully extracting both sets of features, the elements are concatenated to create one single feature vector. This new combined feature vector is provided as input to the anomaly detection module to determine whether an anomaly occurred during this representative time interval.

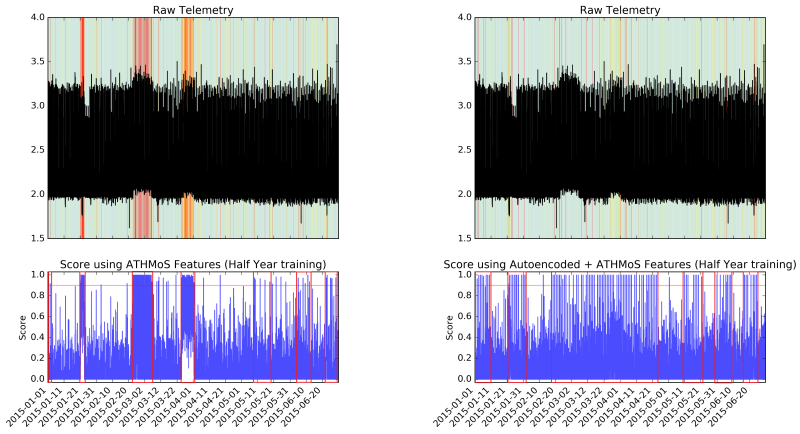
This approach to feature engineering has been used in industry and shown promising results [16, 17]. In our small use-case studies, several interesting conclusions were made. First, it was apparent that by modifying the number of layers and nodes in the autoencoder greatly affected the features and therefore anomalies which are detected. Moreover, even with the exact same network architecture, training models again for even the same parameter resulted in different learned feature vectors - this is due to the random nature in which initial neural network weights are chosen as well as the nature of the network weight/bias updating via gradient descent. Additionally, it was determined that running ATHMoS with only these new 'black-box' features generally resulted in a worse precision in anomaly detection than our existing system with human engineered features. We did however find that found that by including both the autogenerated features *and* our custom engineered ones provided good false-positive rate reduction and the detection of new anomalies. More details on this application can be seen in Figures 9 and 10 and the companion paper [6].



(a) Anomalies Detected Using Default ATHMoS Features

(b) Anomalies Detected Using Default ATHMoS Features Combined with Neural Network Features

Fig. 9 (a) Raw telemetry data (top) and the anomaly scores (bottom) for a given test parameter using the default ATHMoS Features. Note that the thin horizontal red line represents our cut-off value of classifying the telemetry as anomalous whereas the colored shading of the raw telemetry plot indicates the points anomaly score. (b) The anomaly scores after running ATHMoS by combining the default features with the newly obtained features from the neural network. The left peak in plot (a) is falsely flagged as an anomaly using the default ATHMoS features, whereas in (b) the addition of the machine-learned black-box features aid in making a slight adjustment to the overall score and hence making this peak no longer flagged as an anomaly.



(a) Anomalies Detected Using Default ATHMoS Features

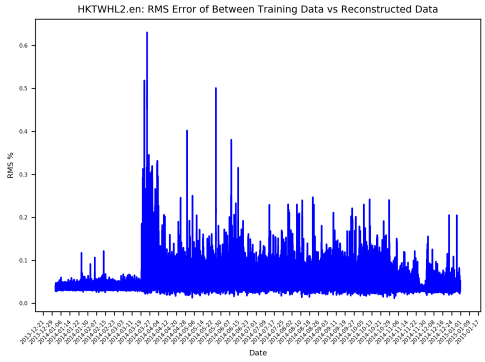
(b) Anomalies Detected Using Default ATHMoS Features Combined with Neural Network Features

Fig. 10 (a) Raw telemetry data (top) and the anomaly scores (bottom) which were correctly detected using the default ATHMoS Features. (b) The anomaly scores after running ATHMoS by combining the default features with the newly obtained features from the neural network. In this example, one sees that the anomalies detected during the months of January, February and March are very distinct (highlighted red box in (a)). Interestingly, the features that were automatically learned by the neural network show that on a finer level of detail, this parameter seemed to fluctuate in a different manner before and after these caught anomaly intervals (red boxes in (b)). This demonstrates that the network had learned some special features of the telemetry which may have indicated the behaviour was behaving non-nominally potentially 2 weeks in advance.

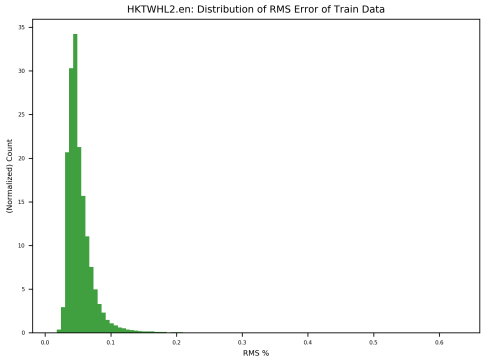
V. Anomaly Detection

In the previous section we described how we use an autoencoder neural network to perform automated feature vector extraction which worked on the basis of inputting sequences of windowed training data and comparing the network output with the same input sequences thereby allowing the network to learn a compressed form of the data. An immediate and intuitive side effect of this learned behaviour is that after training the network and providing new input data, if the data has poor reconstruction accuracy, this implies that the input data was anomalous. See for example, references [18–21] for details on this approach as well as other variants of applying neural networks to this problem.

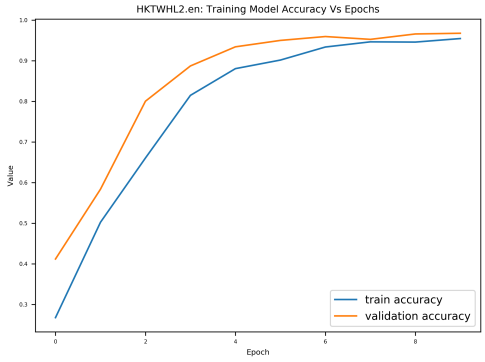
Therefore, by keeping track of the accuracy of each input sequence of training data by computing the root-mean-squared error (RMS-error) between the expected output and actual output we can use a simple standard deviation argument to show that when the RMS-error of a reconstructed sequence of telemetry is 6σ away from the mean RMS-error during training, then this input sequence of data describes an anomaly. We experimentally found that resampling the original raw telemetry data to 5-minute subsamples generally resulted in the best data reconstruction accuracy of approximately 90%. This accuracy was obtained by choosing a neural network node and layer architecture which consisted of 18 input and output nodes and 5 hidden layers consisting of (150, 75, 10, 75, 150) nodes. Figure 11 provides some training network diagnostic plots and Figures 12 and 13 provide an example of some anomaly detection output.



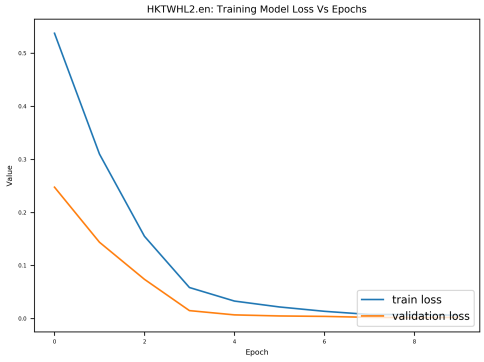
(a) RMS Error for Anomaly Detection Training Data



(b) Normalized Histogram of RMS Error Between Actual and Reconstructed Training Data



(c) Training Accuracy For Each Epoch of Neural Network Learning



(d) Training Loss For Each Epoch of Neural Network Learning

Fig. 11 Various diagnostic plots for a sample telemetry parameter during the neural network training stage for feature vector extraction and anomaly detection. (a) The RMS-error between each sliding window sequence of input data consisting of 18 data points when compared to the networks reconstruction of the same 18 points. (b) After training, this shows the histogram distribution of how well the neural network was able to reconstruct all sliding windows in the input training data. (c) The total training and validation set accuracy after each epoch (a single pass) of the training data during the neural network training. (d) The loss function value of the training and validation sets after each training epoch.

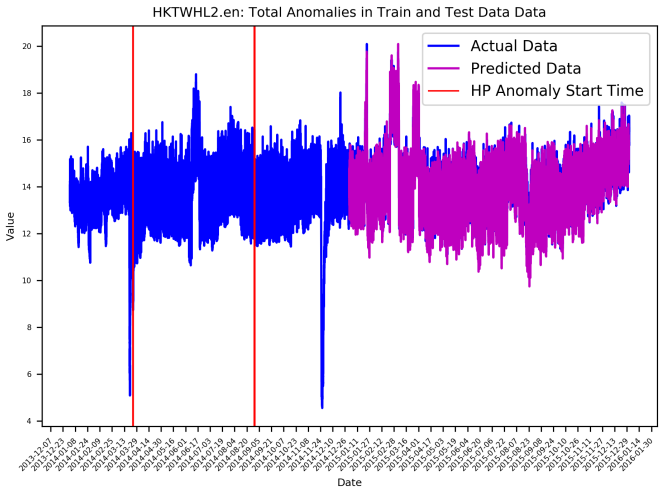


Fig. 12 An example end result of training the neural network with 1 year of training data (2014 in blue), and then running against new test data for all of 2015 (reconstructed data shown in purple). Using only the neural network, no anomalies were detected in the test data although many anomalies are known to have occurred in the months of January, February and March. Two anomaly periods were correctly found in the training data (intervals in red). One of the anomalies detected is shown in detail in Figure 13

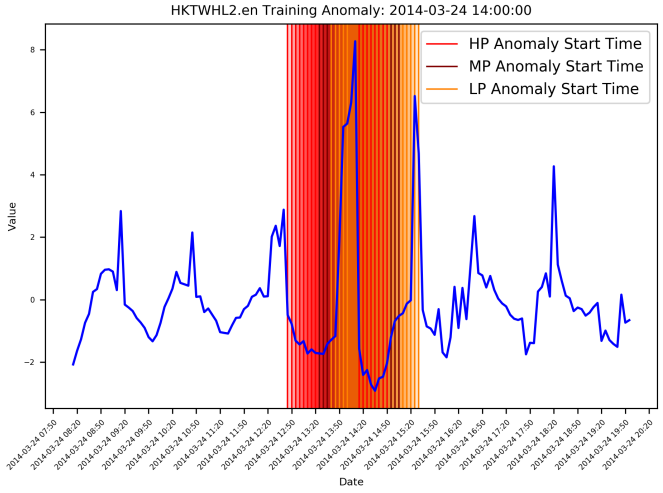


Fig. 13 A correctly identified anomaly in the telemetry parameter HKTWHL2.en detected by the neural network. Each window has a 90-minute length. For each window that has reconstruction error over 6σ away from the mean reconstruction error calculated during training, we start a count of how many of the next 17 neighbouring windows are also above this limit. If more than 6 neighbouring windows (33% of the total orbit) are also over this limit, we assign it to the anomaly class. Additionally, we assign an anomaly priority as either High, Medium or Low depending on the total number of overlapping intervals which are labelled as anomalies. We can see a changing color gradient from when the telemetry was behaving in a High Priority anomaly fashion, then to Medium Priority and finally Low Priority as time increases. Only High Priority anomalies are used in the overall detection calculation when we combine these neural network results with the classic ATHMoS Anomaly detection results. Depending on this High Priority severity, we weigh the anomaly detection in such a way that the results from ATHMoS are favoured, however both results are used in our final anomaly detection decision (ensemble machine learning).

VI. Telemetry and Anomaly Prediction

Another booming area of deep learning is that of time-series forecasting, which has applications to stock market analysis, predictive maintenance and weather forecasting [22–25]. Here we attempted to incorporate the basic principle behind forecasting to satellite telemetry data in order to provide a general identification of the behaviour of telemetry before ATHMoS runs on the new downlinked data once its arrived to our offline processing system.

As an initial step in this direction to prove that such a technology can be useful in space operations we have obtained a simple prediction system which based on the most recent orbit telemetry data (90 minutes) can predict the general trend of telemetry behaviour over the **next 4.5 hours** - the time window in which for a LEO satellite there will likely be another scheduled ground station contact, or, which one can usually be booked.

Starting from a fixed time t, the input sequence of time series data is a 3 element sub-sample of telemetry of the most recent 90 minute interval, $(x_{t-60}, x_{t-30}, x_t)$ and the desired output of the neural network is to predict the next 9 data points which occur at 30 minute intervals $(x_{t+30}, x_{t+60}, x_{t+90}, \ldots x_{t+270})$, thereby predicting data up to 4.5 hours in the future. The training data is created by sliding these training input and output windows over each telemetry parameters historic past data in order to create individual models for each telemetry parameter. This 'look-back horizon' and 'forecast-horizon' concept are depicted in Figure 14 and the neural network architecture in Figure 15.

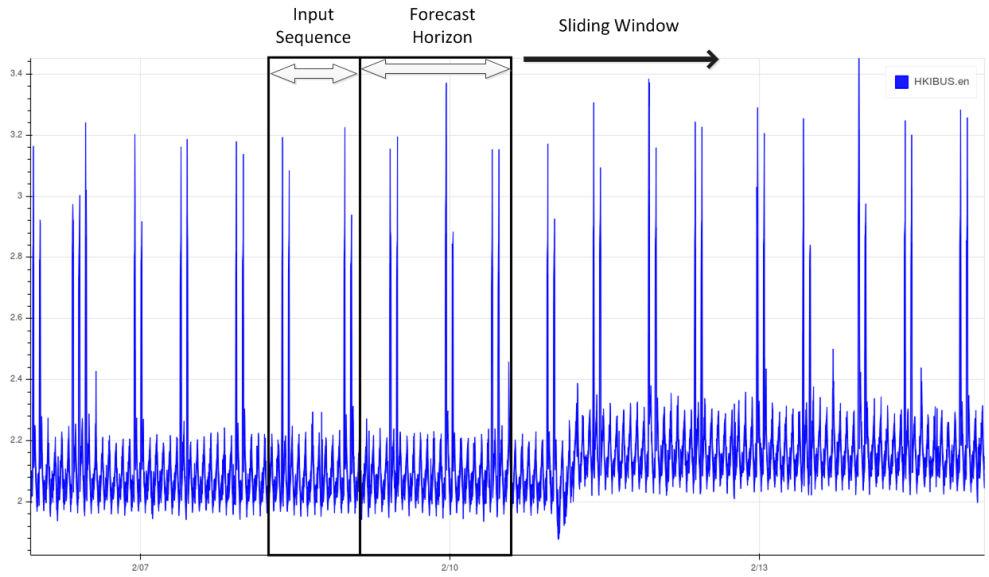


Fig. 14 In this sample plot we see the sliding input window along with its neighbouring forecast horizon window. The forecast horizon window is composed of future telemetry data which we use to train the neural network based on the input sequence of data points.

We found that the network architecture needed to obtain accurate ($\geq$ 75%) training data predictions varied immensely by changing the look-back horizon and/or the forecast horizon sizes. Some key conclusions we reached in designing a network architecture are:

1) A larger look-back horizon period as input to the network requires more layers and/or nodes needed to be added in order to compensate for the larger variance between a single input sequence.
2) A larger forecast horizon requires a sufficiently large look-back horizon since the future behaviour needs to depend on a certain complexity of the input data.
3) Too fine grained (small time steps) look-back and forecast horizons required such a complicated network structure ($\geq$4 hidden layers and/or hundreds of nodes) that the network needed large amounts of training data (years), number of epochs ($\sim$ 1000) and running time ($\sim$ 6 hours per parameter) to obtain an end result. Due to the nature/art of designing network architectures, this approach was not feasible.

To get a general idea of how the accuracy of the predictions can change based on the number of hidden layers and nodes in the network, Figure 16 shows the accuracy difference in training data predictions based on two different numbers of hidden layer nodes.

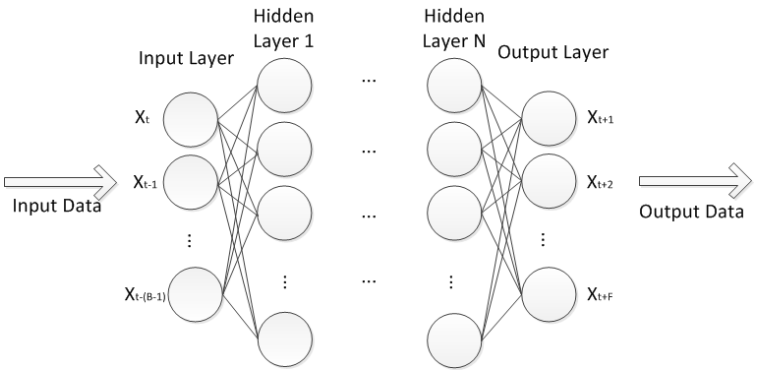
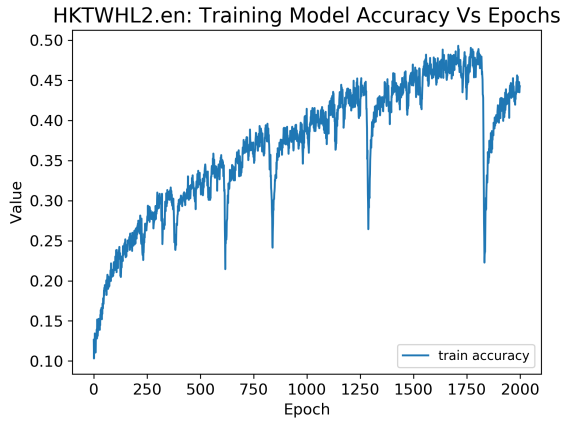
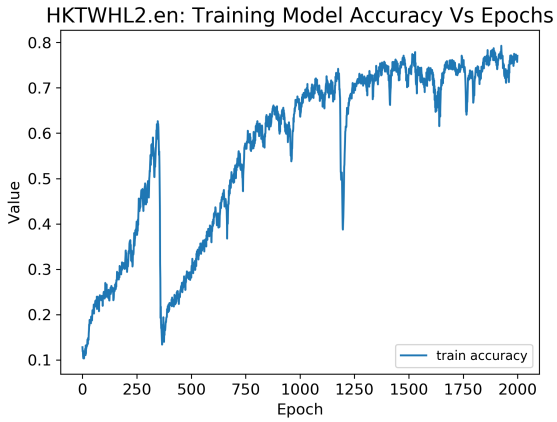


Fig. 15 A neural network architecture for time series prediction. The input layer takes a sequence of time series data as input $(x_{t-(B-1)}, \ldots, x_{t-1}, x_t)$ where B is the look-back number (the number of time steps to increment), and the output layer provides prediction of time series values $(x_{t+1}, x_{t+2}, \ldots, x_{t+F})$, where $F \geq 2$ is the forecast number. A main result of this experimental module that was the number of look-back and forecast time steps depends greatly on one another and the number of hidden nodes and layers varies significantly on these parameters. We found it difficult to create a general neural network architecture which works for multiple telemetry parameters individually if the sample rate of the input data was too high, e.g. 1 data point per 30 seconds. Instead, we opted to use as input 3 data points separated by 30-minute intervals (corresponding to a LEO orbit) and then forecast the next nine 30-minute time steps ($F = 9$). This therefore predicts the general behaviour of the data for the next 4.5 hours.



(a) (75, 75) Hidden Layer Architecture Training Accuracy



(b) (250, 250) Hidden Layer Architecture Training Accuracy

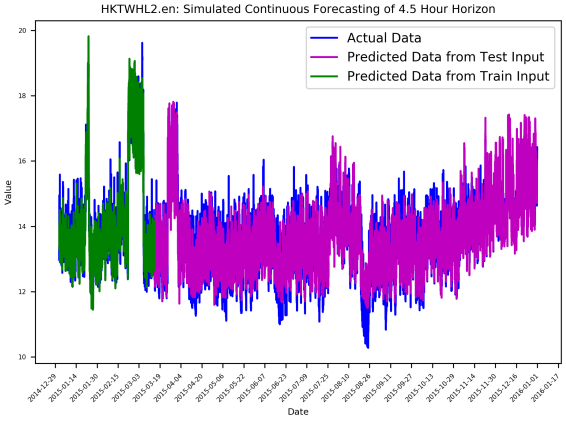
Fig. 16 An accuracy comparison between different hidden layer number of nodes for our time series forecasting neural network. It is clear that increasing the number of nodes in both hidden layers allows the network to learn more of the inherent patterns in the data due to the increased variance of the model.

As an initial starting point for our implementation, we attempted to use 8-12 months of past data to train each telemetry parameter. In the end, this was not successful due to the fact that the networks would not be able to learn the complicated behaviour which resulted from the many states the data took throughout the year. Instead, we determined it was better to confine the training data to smaller time intervals (1.5-3 months) where the telemetry behaved relatively nominally. That is, having anomalies or large state jumps in the telemetry parameters in a short time span allowed the network to not overtrain or underfit the input data, as opposed to it trying to learn 'too many things at once'. The final network architecture which we found gave the most promising results for forecasting multiple telemetry parameters was of the form $(250, 250)$. See Figure 16(b) for a sample training accuracy plot.

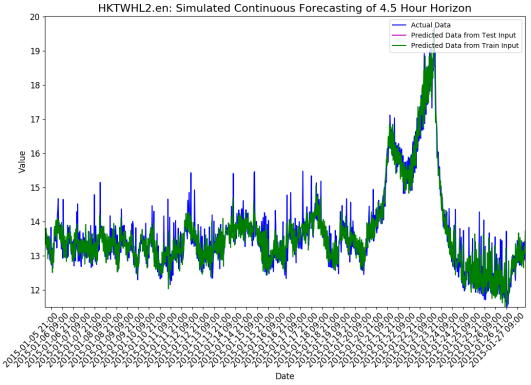
An example of an approximate 3 month training period and 9 month continually running test period is provided in

Figure 17. There we can see that within the training data, several states of data occur and additionally, there are two large periods of time where anomalies occurred. In the remaining 9 months of the year, the general telemetry behavior was successfully forecasted. For some telemetry parameters we determined if the initial training data did not include several states of behavior, the prediction accuracy decreased as the year progressed due to the (intuitive) fact that the training model did not know what to do with new telemetry patterns. Therefore, in the ongoing implementation of this application, we are adding a model update loop which re-trains the existing model every 2 weeks to include the most recent data. In this sense, the neural network is continually learning the new telemetry behavior and hence can more accurately perform the forecasting.

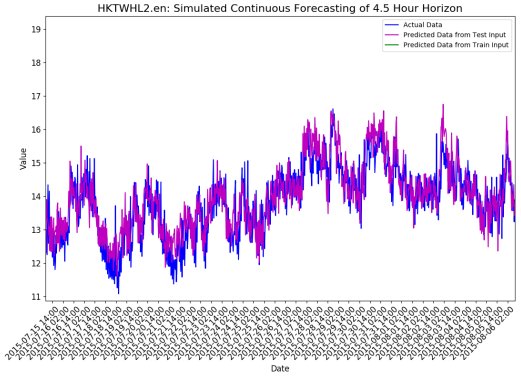
The connection to ATHMoS and anomaly prediction is now clear. After obtaining these telemetry predictions, we are able to feed the predicted telemetry into our existing ATHMoS system to analyse whether an anomaly will occur. The general workflow which is being developed is defined by Figure 18 where we flag any potential anomalies as forecasted by the neural network module and set an interval flag on this parameter for the next 6 hours. When actual new data is downlinked, we run the typical ATHMoS anomaly detection on the new data and assign a higher priority to any detected anomaly which indicates that it was also correctly pre-detected.



(a) 1 Year of Forecasting Predictions



(b) Subsample of 3 Weeks Training Data Predictions



(c) Subsample of 3 Weeks Test Data Predictions

Fig. 17 (a) 4.5 hour predictions of training data (green), test data (purple) against actual raw data (blue) for a simulated run of an entire year (2015). The neural network correctly predicts the future data throughout the training and test periods including anomalous time periods. (b) A zoomed-in plot showing 4.5 hour predictions sequentially windowed and displayed in green (training data) against actual data (blue). Here we see the network learned how to predict the sharp anomaly peak as well as filter the noisy data points. (c) A zoomed-in plot of new test data predictions for 3 weeks of simulated runtime.

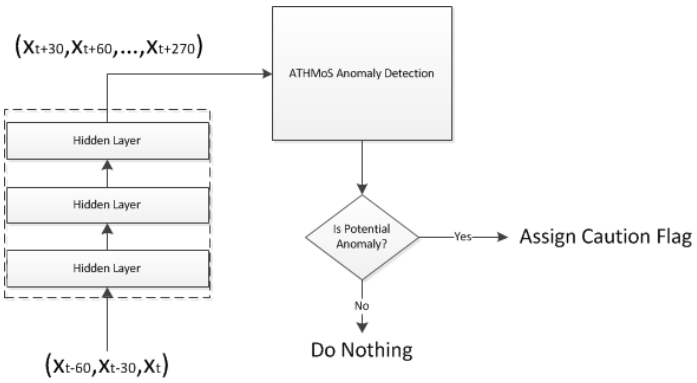


Fig. 18 The general workflow of anomaly prediction currently being implemented at GSOC. Past telemetry data is fed into the neural network module (dashed box) which then produces telemetry forecasting. This output is passed to the anomaly detection module where we determine whether the predicted data is anomalous. If we determine it could be a potential anomaly, the corresponding future time interval is flagged in which case we await later (more recent) telemetry to be analysed. If a real anomaly is detected, we add an additional classification to it which indicates that the recent telemetry data had showed signs of non-nominal behavior.

VII. Conclusion

In this paper, we discussed the results of a year of research into three different applications of deep learning neural networks to analyse satellite telemetry data. After initial experiments and iterative algorithm changes, two different types of neural networks were eventually created: an autoencoder neural network for feature vector extraction and anomaly detection, and a multi-layer LSTM network for telemetry forecasting and anomaly prediction. We have outlined the architectures of these two networks and discussed how we are currently implementing them into ATHMoS in order to augment the existing system by detecting different types of anomalies and reducing false positive flags.

The general conclusion that the authors reached was that neural networks are powerful tools which can be used for many interesting applications in satellite telemetry monitoring - however, this comes at a cost. The black-box nature of the networks made it very difficult to correctly design networks and obtain accurate results, albeit when a correct approach was determined the results were often remarkable (e.g. Figure 17 showing consecutive 4.5 hour future predictions and noise filtration). Furthermore, the inherent “magic" of certain applications such as the unsupervised automatic feature extraction provides results which are not human interpretable or understandable. For example, sometimes it would happen that an anomaly was detected using these features, however we found it difficult to manually inspect the raw telemetry to determine whether it was a false positive, or, if the system really learned something deeper. Since the end users of our product are engineers, we want them to be able to trust and understand the results from a normal human and mathematical perspective. Therefore, another important conclusion was made that for all of our applications, we are careful to weigh the neural network detections and predictions and provide a higher dependence on the results from our existing software ATHMoS wherever possible.

This initial study was meant to shed light on the promise of using deep learning in space operations and we feel that there will be great progress in this area in the future. Most machine learning algorithms in other industries are being replaced and/or augmented with deep learning due to the improved accuracy and performance. Since big data and machine learning have only recently been more adopted to aid in space operations, we see this study as a next step towards the future of using Artificial Intelligence in our field.

Acknowledgements

We gratefully acknowledge Christian Reinhardt, Nikolas Pomplun and Arvind Kumar Balan for useful discussions.

References

[1] OMeara, C., Schlag, L., Faltenbacher, L., and Wickler, M., "ATHMoS: Automated Telemetry Health Monitoring System at GSOC using Outlier Detection and Supervised Machine Learning," *14th International Conference on Space Operations*, pp. 2347, 2016.

[2] http://cassandra.apache.org

[3] http://spark.apache.org

[4] https://keras.io/

[5] https://www.tensorflow.org/

[6] Schlag, L., OMeara, C., and Wickler, M., "On Evaluation of Outlier Rankings and Outlier Scores," *15th International Conference on Space Operations*, 2018.

[7] LeCun, Y., Bengio, Y., and Hinton, G., "Deep Learning," *Nature*, Vol. 521, No. 7553, pp. 436, 2015.

[8] Schmidhuber, J., "Deep Learning in Neural Networks: An Overview," *Neural Networks*, Vol. 61, pp. 85-117, 2015.

[9] Hochreiter, S., and Schmidhuber, J., "Long Short-Term Memory," *Neural Computation*, Vol. 9, No. 8, pp. 1735-1780, 1997.

[10] Hecht-Nielsen, R., "Theory of the Backpropagation Neural Network," *Neural Networks for Perception*, pp. 65-93, 1992.

[11] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R., "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *The Journal of Machine Learning Research*, Vol. 15, No. 1, pp. 1929-1958, 2014.

[12] Hinton, G.E., and Salakhutdinov, R.R., "Reducing the Dimensionality of Data with Neural Networks," *Science*, Vol. 313, No. 5786, pp. 504-507, 2006.

[13] Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P. A., "Extracting and Composing Robust Features with Denoising Autoencoders," *Proceedings of the 25th International Conference on Machine Learning*, pp. 1096-1103, 2008.

[14] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P. A. "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion," *Journal of Machine Learning Research*, pp. 3371-3408, 2010.

[15] Coates, A., Ng, A., and Lee, H., "An Analysis of Single-Layer Networks in Unsupervised Feature Learning," *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011.

[16] Laptev, N., Yosinski, J., Li, E., and Slawek, S., "Time-series Extreme Event Forecasting with Neural Networks at Uber," *ICML Time Series Workshop*, 2017.

[17] Zhu, L., and Laptev, N., "Deep and Confident Prediction for Time Series at Uber," *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pp. 103-110, 2017.

[18] Martinelli, M., Tronci, E., Dipoppa, G. and Balducelli, C., "Electric Power System Anomaly Detection Using Neural Networks," *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pp. 1242-1248, 2004.

[19] Sakurada, M. and Yairi, T., "Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction," *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, pp. 4, 2014.

[20] Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P. and Shroff, G., "LSTM-based Encoder-decoder for Multi-sensor Anomaly Detection," *arXiv preprint arXiv:1607.00148*, 2016.

[21] Shipmon, D. T., Gurevitch, J. M., Piselli, P. M., and Edwards, S. T., "Time Series Anomaly Detection; Detection of Anomalous Drops with Limited Features and Sparse Examples in Noisy Highly Periodic Data," *arXiv preprint arXiv:1708.03665*, 2017.

[22] Azoff, E. M., "Neural Network Time Series Forecasting of Financial Markets," John Wiley & Sons, Inc., 1994.

[23] Hill, T., O'Connor, M., and Remus, W., "Neural Network Models for Time Series Forecasts," *Management Science*, Vol. 42, No. 7, pp. 1082-1092, 1996.

[24] Zhang, G. P., and Qi, M., "Neural Network Forecasting for Seasonal and Trend Time Series," *European Journal of Operational Research*, Vol. 160, No. 2, pp. 501-514, 2005.

[25] Xingjian, S.H.I., Chen, Z., Wang, H., Yeung, D.Y., Wong, W.K. and Woo, W.C., "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting," *Advances in Neural Information Processing Systems*, pp. 802-810, 2015.